Recent Advances in Integrating

Machine Learning and Combinatorial Optimization

Elias B. Khalil

Andrea Lodi

Didier Chételat

Antoine Prouvost

35th AAAI Conference

Bistra Dilkina

Maxime Gasse

Giulia Zarpellon

Laurent Charlin

CANADA EXCELLENCE RESEARCH CHAIR
DATA SCIENCE FOR REAL-TIME DECISION-MAKING
UNIVERSITY OF TORONTO
POLYTECHNIQUE MONTRÉAL
USC University of Southern California

Part 2

The pure ML approach: predicting feasible solutions

Elias Khalil, Andrea Lodi, Bistra Dilkina, Didier Chételat,
Maxime Gasse, Antoine Prouvost, Giulia Zarpellon, Laurent Charlin

MH7: Recent Advances in Integrating
Machine Learning and Combinatorial Optimization

35th AAAI Conference
on Artificial Intelligence
February 3rd 2021

Towards Tailored Algorithms

IBM Knowledge Center

- Managing sets of parameters
- Parameter names
- Correspondence of parameters between APIs
- Saving parameter settings to a file in the C API
- **Topical list of parameters**
  - Barrier
  - Benders algorithm
  - Distributed MIP
  - **MIP**
    - MIP general
    - **MIP strategies**
    - MIP cuts
    - MIP tolerances
    - MIP limits

Here are links to parameters controlling MIP strategies.

algorithm for initial MIP relaxation

Benders strategy

MIP subproblem algorithm

MIP variable selection strategy

MIP strategy best bound interval

MIP branching direction

backtracking tolerance

MIP dive strategy

MIP heuristic effort

CPLEX Documentation

Towards Tailored Algorithms

MIP variable selection strategy

Value	Symbol	Meaning
-1	CPX_VARSEL_MININFEAS	Branch on variable with minimum infeasibility
0	CPX_VARSEL_DEFAULT	Automatic: let CPLEX choose variable to branch on; **default**
1	CPX_VARSEL_MAXINFEAS	Branch on variable with maximum infeasibility
2	CPX_VARSEL_PSEUDO	Branch based on pseudo costs
3	CPX_VARSEL_STRONG	Strong branching
4	CPX_VARSEL_PSEUDOREDUCED	Branch based on pseudo reduced costs

MIP heuristic frequency

Value	Meaning
-1	None
0	Automatic: let CPLEX choose; **default**
Any positive integer	Apply the periodic heuristic at this frequency

CPLEX Documentation

Towards Tailored Algorithms

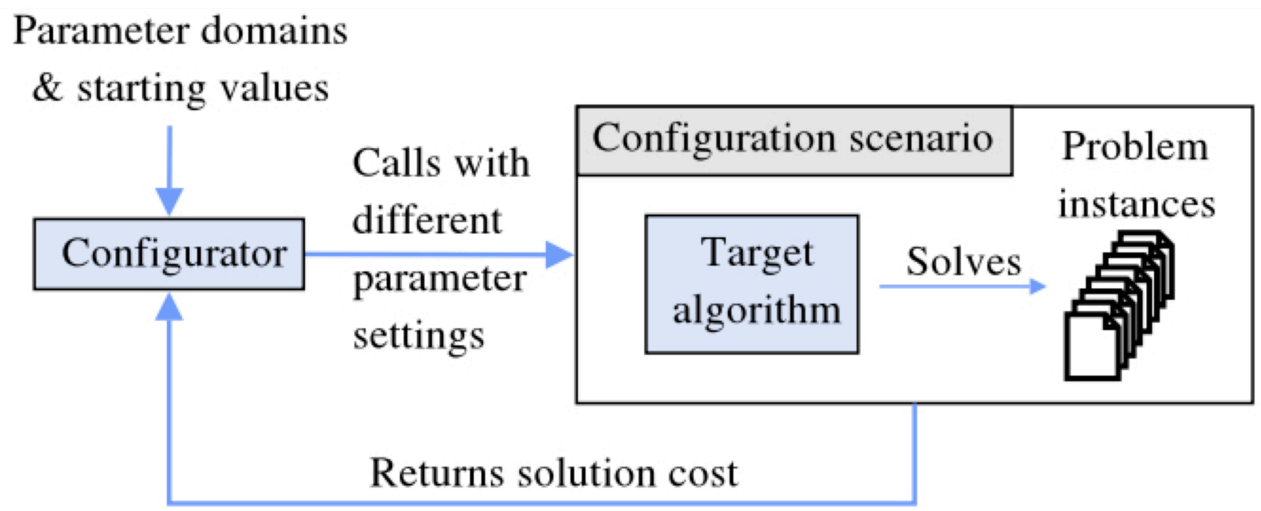


Fig. 1. A configuration procedure (short: configurator) executes the target algorithm with specified parameter settings on one or more problem instances, observes algorithm performance, and uses this information to decide which subsequent target algorithm runs to perform. A configuration scenario includes the target algorithm to be configured and a collection of instances.

Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown. "Automated configuration of mixed integer programming solvers." CPAIOR, 2010.

Towards Tailored Algorithms

See IJCAI-20 Tutorial: https://www.automl.org/tutorial_ac_ijcai20/

Automated Algorithm Configuration

- ParamILS [Hutter et al., JAIR 2009], SMAC [Hutter et al., LION 2011]
- Key Idea: search over parameter configurations
  - Stochastic Local Search or Bayesian Optimization

- Great for algorithms with many parameters
  - 2-52x speedups for CPLEX on some problem distributions [Hutter et al., CPAIOR 2010]

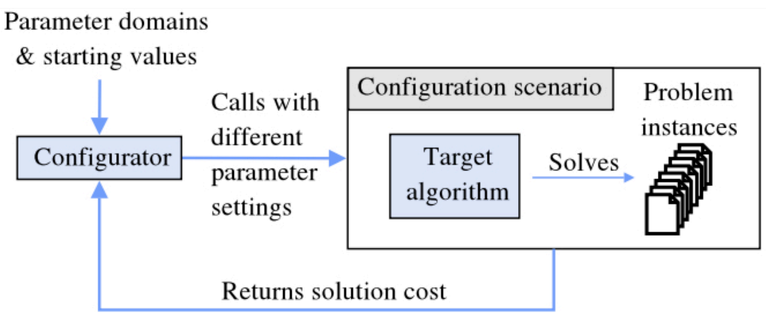


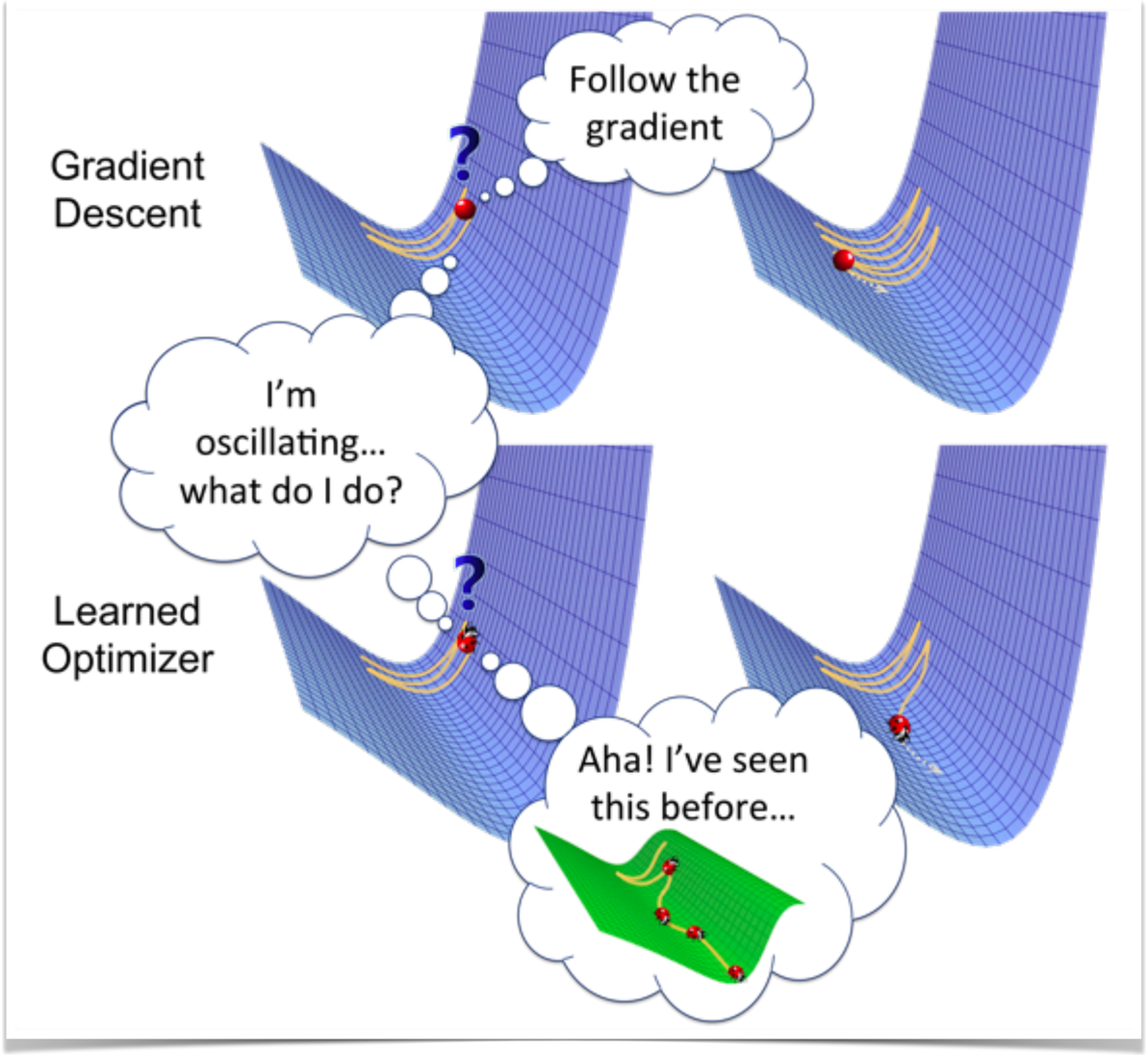
Fig. 1. A configuration procedure (short: configurator) executes the target algorithm with specified parameter settings on one or more problem instances, observes algorithm performance, and uses this information to decide which subsequent target algorithm runs to perform. A configuration scenario includes the target algorithm to be configured and a collection of instances.

Limitations

- Operates at the instance-level, not the **algorithm iteration-level**
- Assumes human-designed parameter space is rich enough

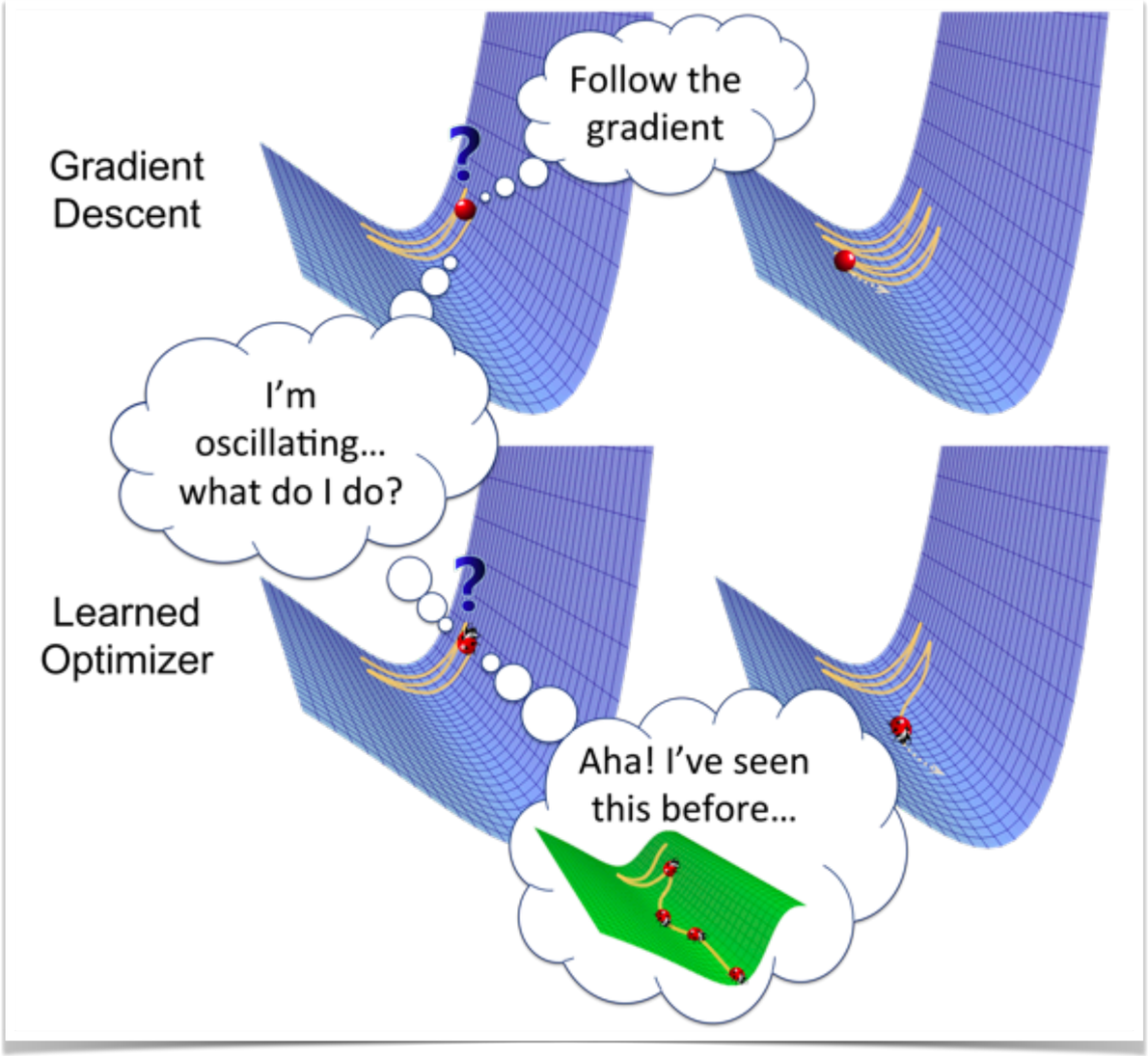
Warm-up: Learning in Gradient Descent

Source: Ke Li, http://bair.berkeley.edu/blog/2017/09/12/learning-to-optimize-with-rl
Li, Ke, and Jitendra Malik. "Learning to optimize." arXiv:1606.01885, 2016 and ICLR, 2017



Warm-up: Learning in Gradient Descent

Source: Ke Li, http://bair.berkeley.edu/blog/2017/09/12/learning-to-optimize-with-rl
Li, Ke, and Jitendra Malik. "Learning to optimize." arXiv:1606.01885, 2016 and ICLR, 2017



Algorithm 1 General structure of optimization algorithms

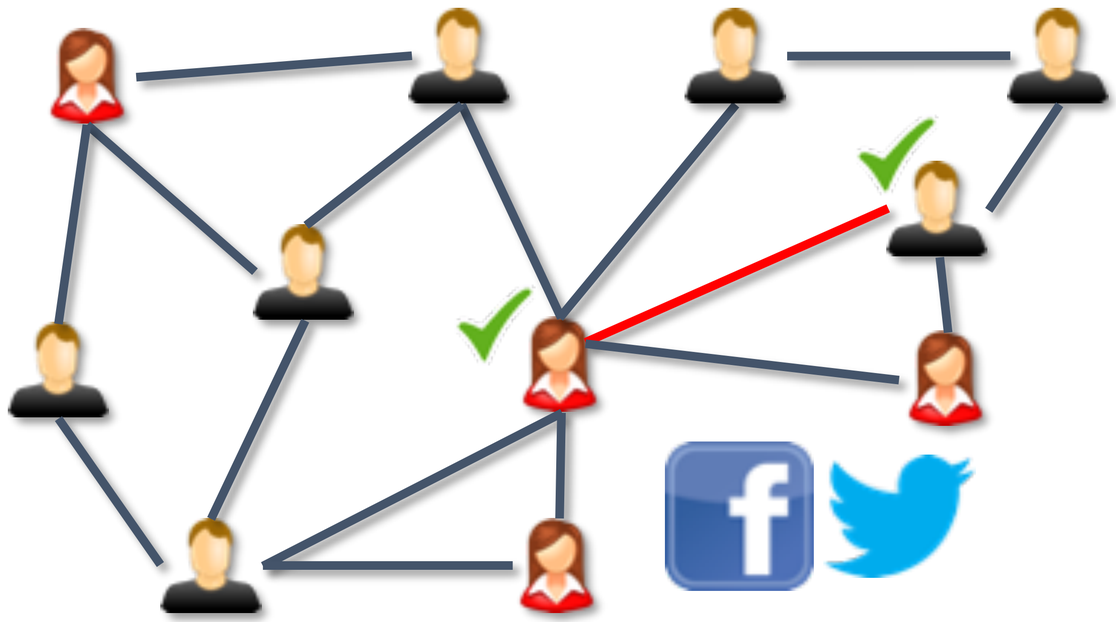
Require: Objective function f
$x^{(0)} \leftarrow$ random point in the domain of f
for $i = 1, 2, \ldots$ **do**
$\Delta x \leftarrow \boxed{\phi(\{x^{(j)}, f(x^{(j)}), \nabla f(x^{(j)})\}_{j=0}^{i-1})}$
if stopping condition is met **then**
return $x^{(i-1)}$
end if
$x^{(i)} \leftarrow x^{(i-1)} + \Delta x$
end for

Gradient Descent	$\phi(\cdot) = -\gamma \nabla f(x^{(i-1)})$
Momentum	$\phi(\cdot) = -\gamma \left(\sum_{j=0}^{i-1} \alpha^{i-1-j} \nabla f(x^{(j)}) \right)$
Learned Algorithm	$\phi(\cdot) =$ Neural Net

Greedy Graph Optimization

Minimum Vertex Cover
Find smallest vertex subset such that each edge is covered

2-Approximation:
Greedily add vertices of edge with **max degree sum**



Learning Combinatorial Optimization Algorithms over Graphs

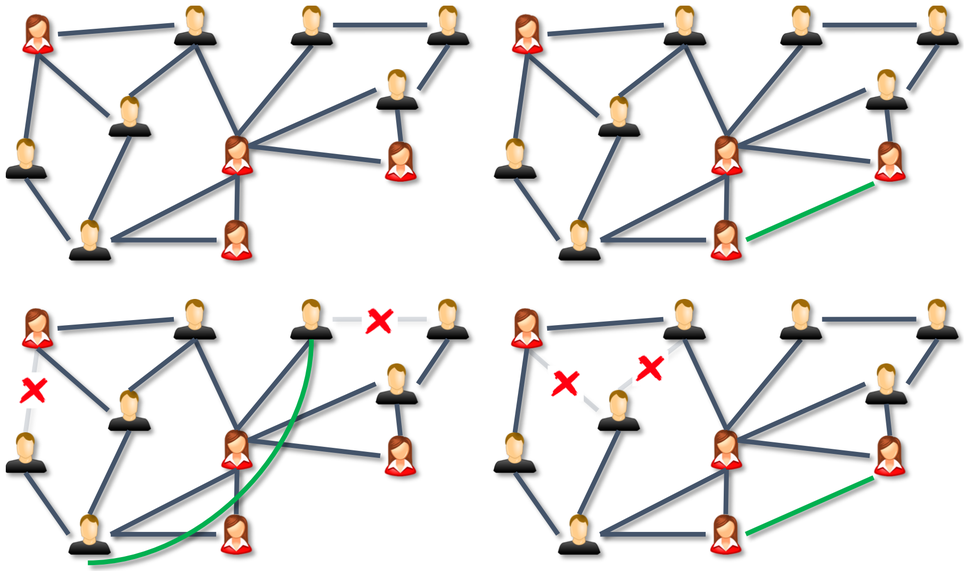
Hanjun Dai[†*], Elias B. Khalil[†*], Yuyu Zhang[†], Bistra Dilkina[†], Le Song[†§]
[†] College of Computing, Georgia Institute of Technology
[§] Ant Financial
{hanjun.dai, elias.khalil, yuyu.zhang, bdilkina, lsong}@cc.gatech.edu

NeurIPS 2017

Problem Statement

Dai & Khalil et al., Learning Combinatorial Optimization Algorithms over Graphs. NeurIPS 2017.

Given a **graph optimization problem G** and a **distribution $\mathcal{D}$** of problem instances, can we **learn better greedy heuristics** that generalize to unseen instances from $\mathcal{D}$?

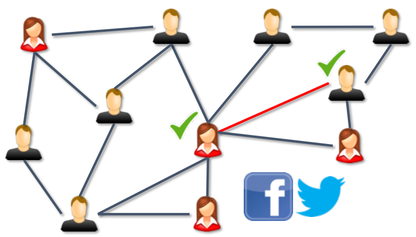


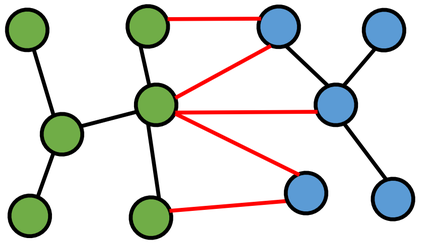
$\sim \mathcal{D}$

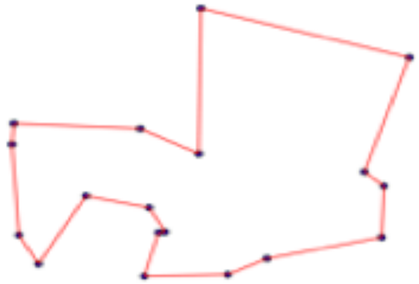
Learning Greedy Heuristics

Given: graph problem, family of graphs
Learn: a **scoring function** to **guide** a **greedy** algorithm

Problem	Minimum Vertex Cover	Maximum Cut	Traveling Salesman Problem
Domain	Social network snapshots	Spin glass models	Package delivery
Greedy operation	Insert nodes into cover	Insert nodes into subset	Insert nodes into sub-tour





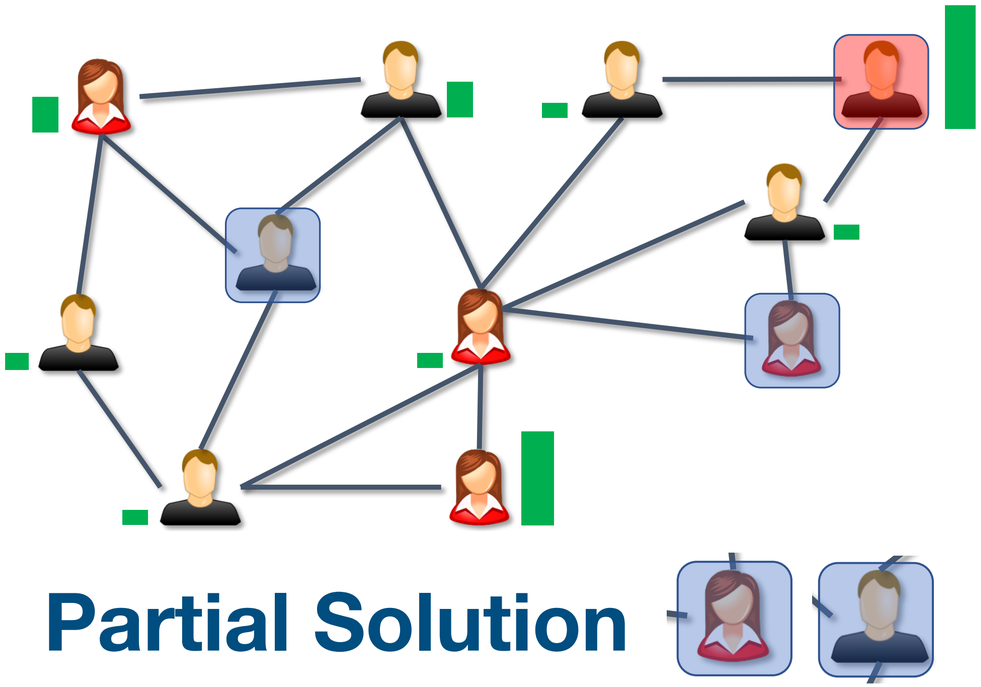


Reinforcement Learning

Greedy Algorithm		Reinforcement Learning
Partial solution	≡	State
Scoring function	≡	Q-function
Select best node	≡	Greedy Policy

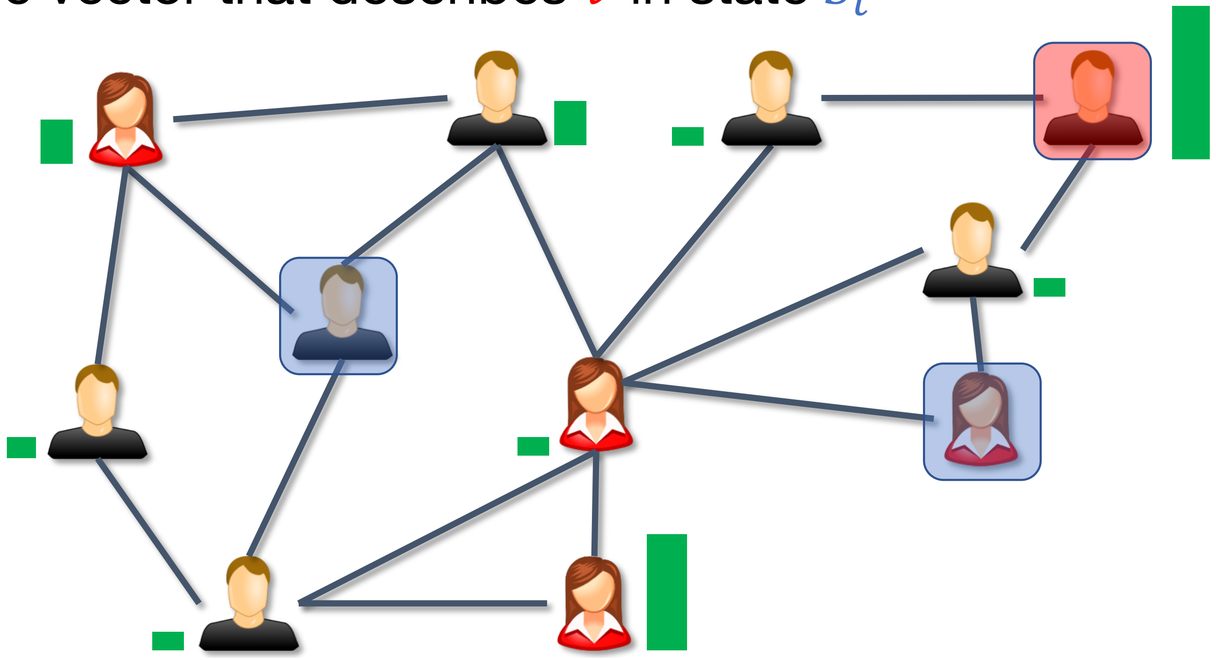
Repeat until all edges are covered:

1. **Compute** node scores
2. **Select** best node w.r.t. score
3. **Add** best node to partial sol.



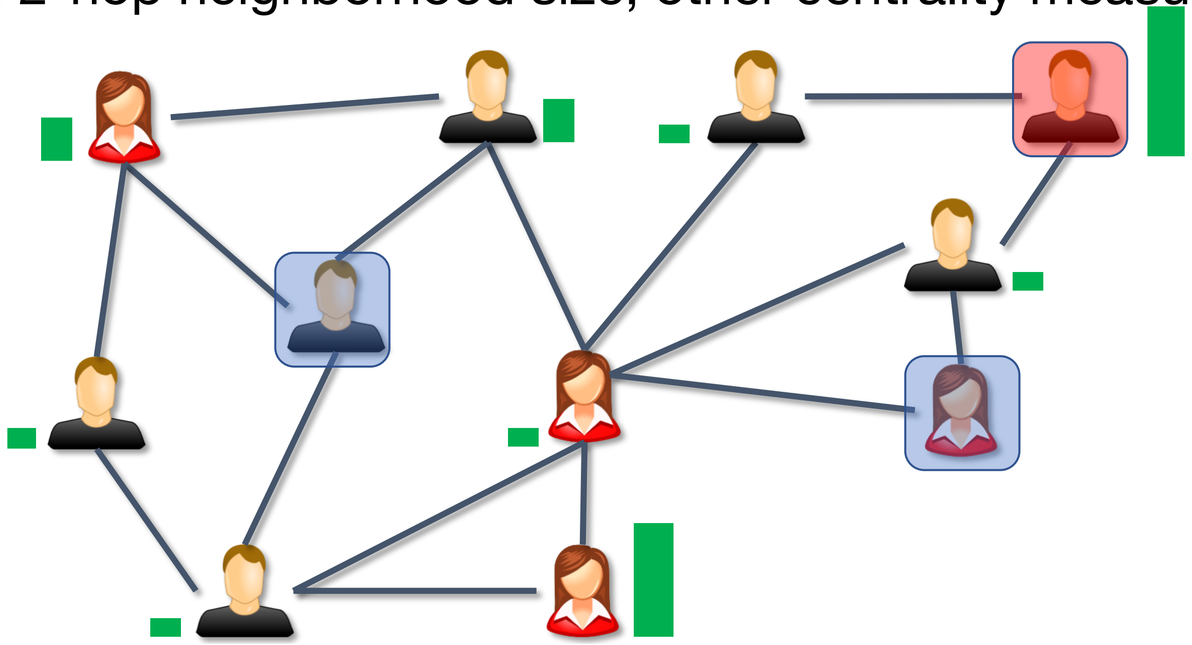
Representing Nodes

- **Action value function**: $\hat{Q}(S_t, v; \Theta)$
  - Estimate of goodness of vertex v in state S_t
- **Representation of v**
  - A feature vector that describes v in state S_t



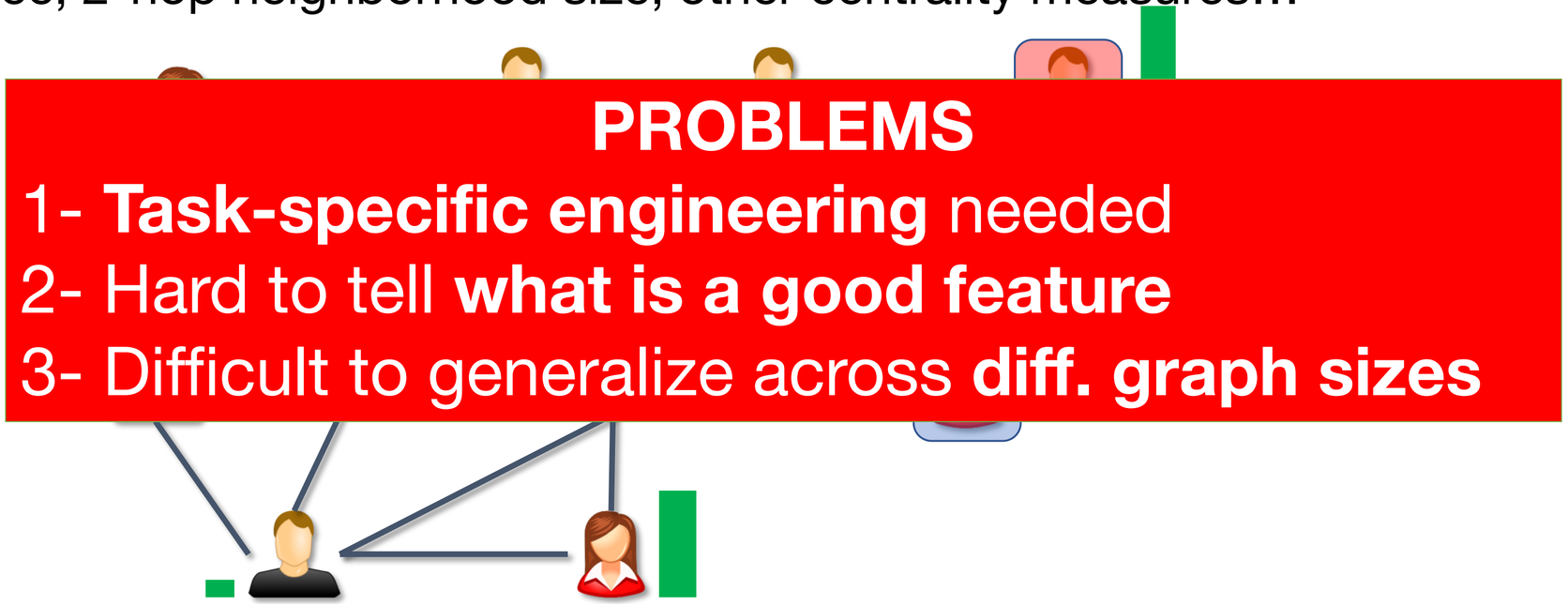
Representing Nodes

- **Action value function**: $\hat{Q}(S_t, v; \Theta)$
  - Estimate of goodness of vertex v in state S_t
- **Representation of v: Feature engineering**
  - Degree, 2-hop neighborhood size, other centrality measures…



Representing Nodes

- **Action value function**: $\hat{Q}(S_t, v; \Theta)$
  - Estimate of goodness of vertex v in state S_t
- **Representation of v: Feature engineering**
  - Degree, 2-hop neighborhood size, other centrality measures…



PROBLEMS

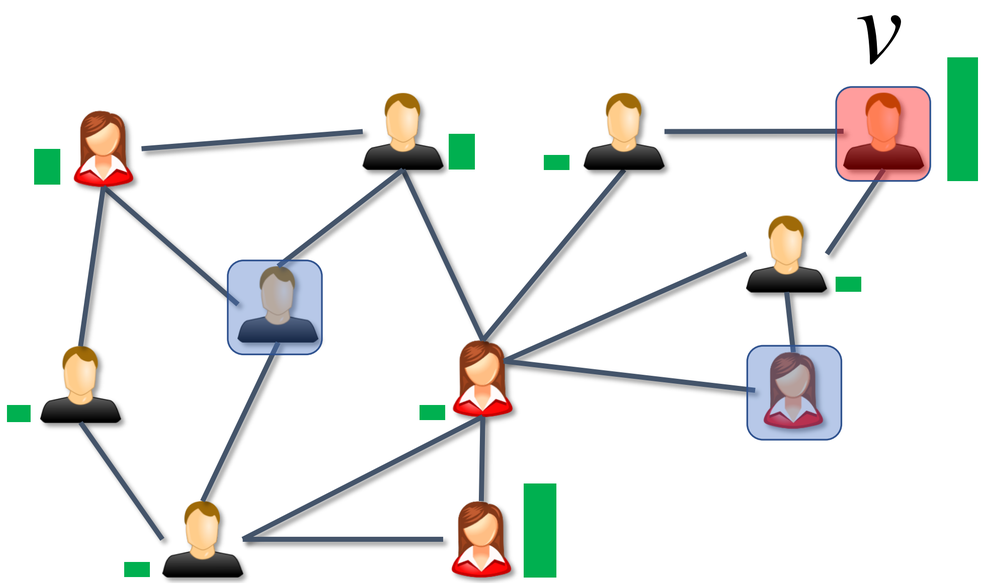
1- **Task-specific engineering** needed

2- Hard to tell **what is a good feature**

3- Difficult to generalize across **diff. graph sizes**

Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first



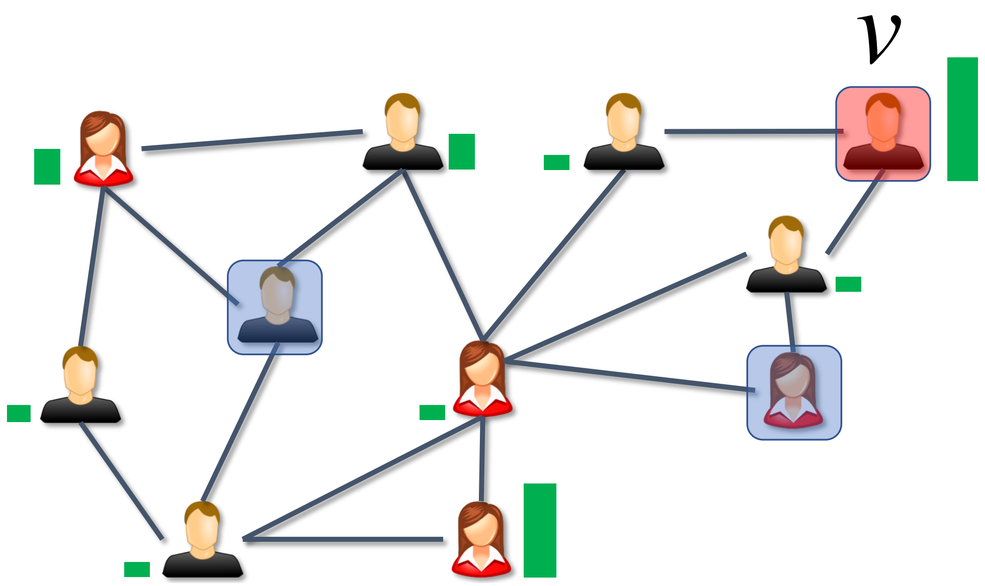
$$v = Q(S_t, v; \Theta)$$

$S_t = \{$ $\}$

Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first

Problem: Not clear what good node features are!



$$= Q(S_t, v; \Theta)$$

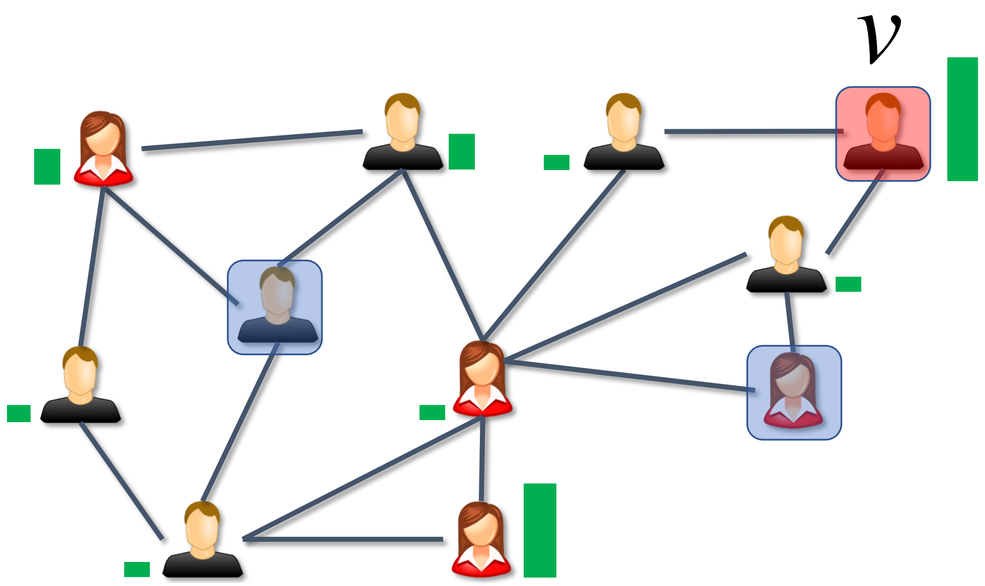
$S_t = \{$ $\}$

Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first

Problem: Not clear what good node features are!

Solution: Parametrize a **Graph Neural Network** with parameters Θ



$$= Q(S_t, v; \Theta)$$

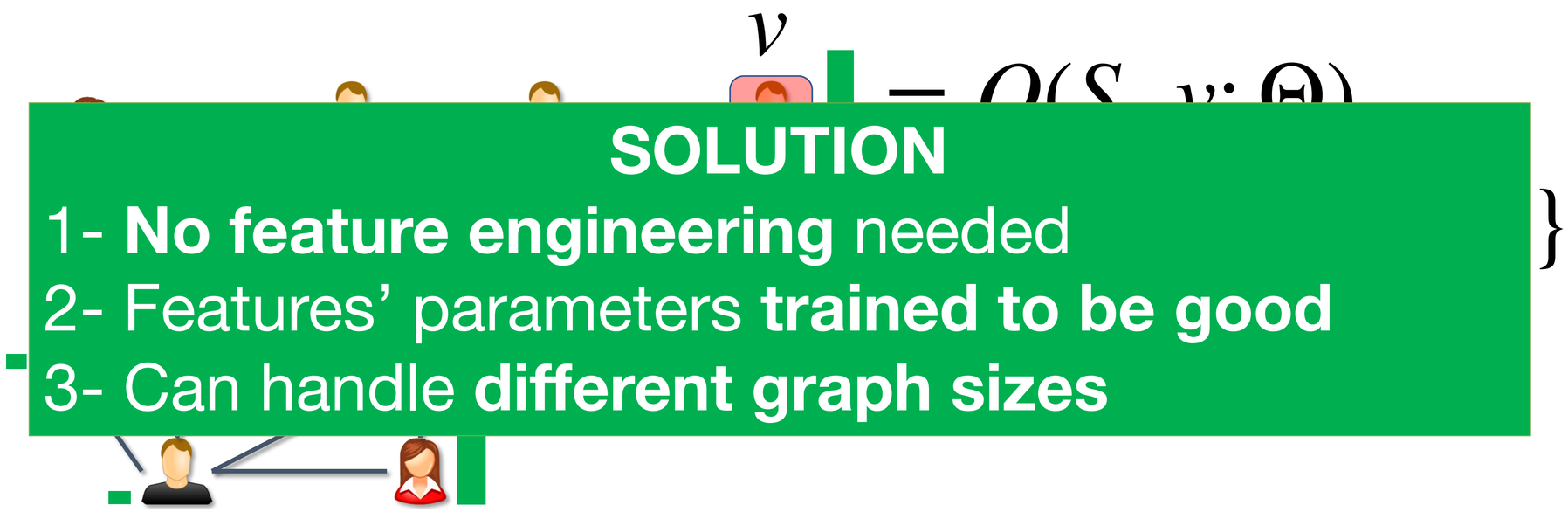
$$S_t = \{$$ $$\}$$

Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first

Problem: Not clear what good node features are!

Solution: Parametrize a **Graph Neural Network** with parameters Θ



SOLUTION

1- **No feature engineering** needed
2- Features' parameters **trained to be good**
3- Can handle **different graph sizes**

Reinforcement Learning Algorithm

Algorithm 1 Q-learning for the Greedy Algorithm

Initialize experience replay memory $\mathcal{M}$ to capacity N
for episode $e = 1$ **to** L **do**
  Draw graph G from distribution $\mathbb{D}$
  Initialize the state to empty $S_1 = ()$
  for step $t = 1$ **to** T **do**
    $v_t = \begin{cases} \text{random node } v \in \overline{S}_t, & \text{w.p. } \epsilon \\ \text{argmax}_{v \in \overline{S}_t} \widehat{Q}(h(S_t), v; \Theta), & \text{otherwise} \end{cases}$
    Add v_t to partial solution: $S_{t+1} := (S_t, v_t)$
    if $t \geq n$ **then**
      Add tuple $(S_{t-n}, v_{t-n}, R_{t-n,t}, S_t)$ to $\mathcal{M}$
      Sample random batch from $B \overset{iid.}{\sim} \mathcal{M}$
      Update Θ by SGD over (6) for B
    end if
  end for
end for
return Θ

Θ: model parameters

Depend on vertex features

Sample graph instance

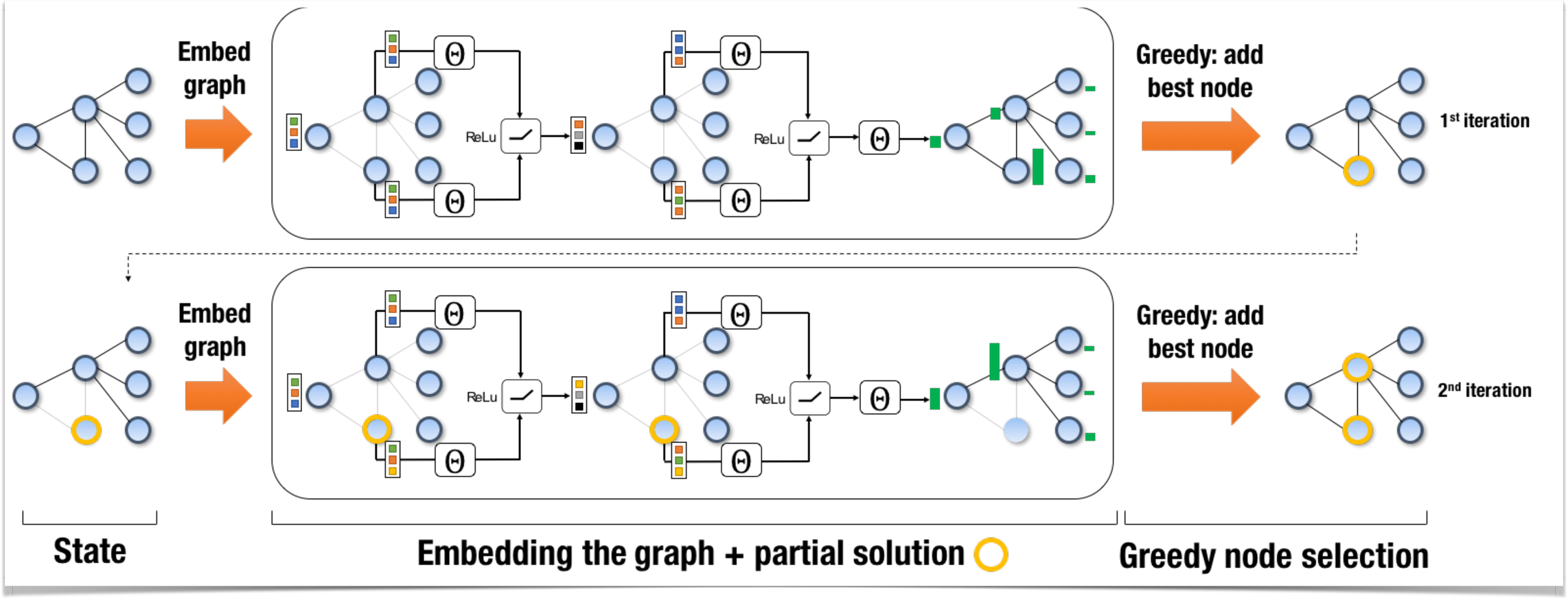
Explore or
Exploit according to current policy

Update state

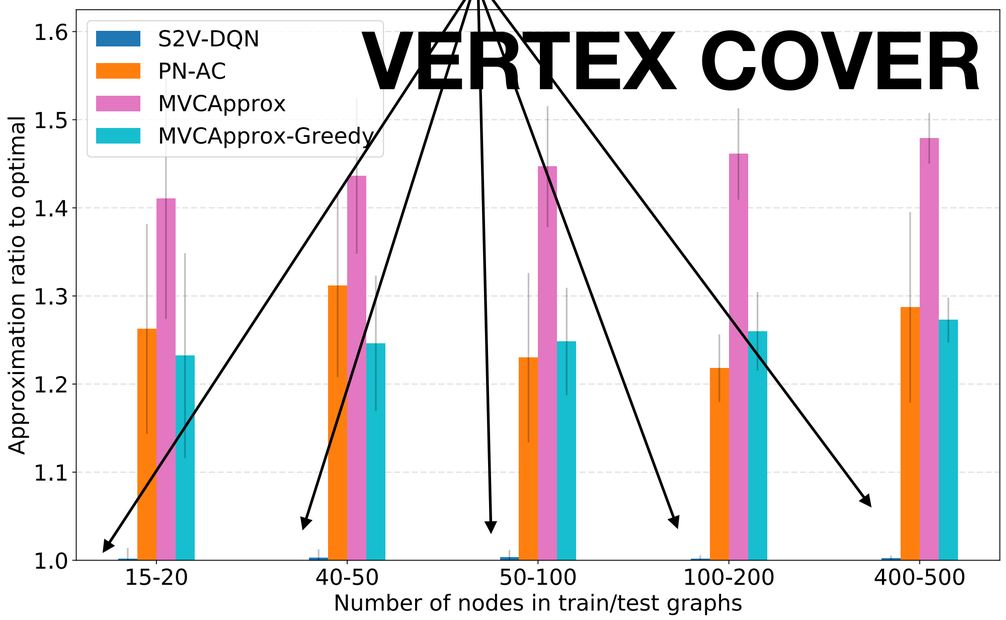
Optimize model parameters

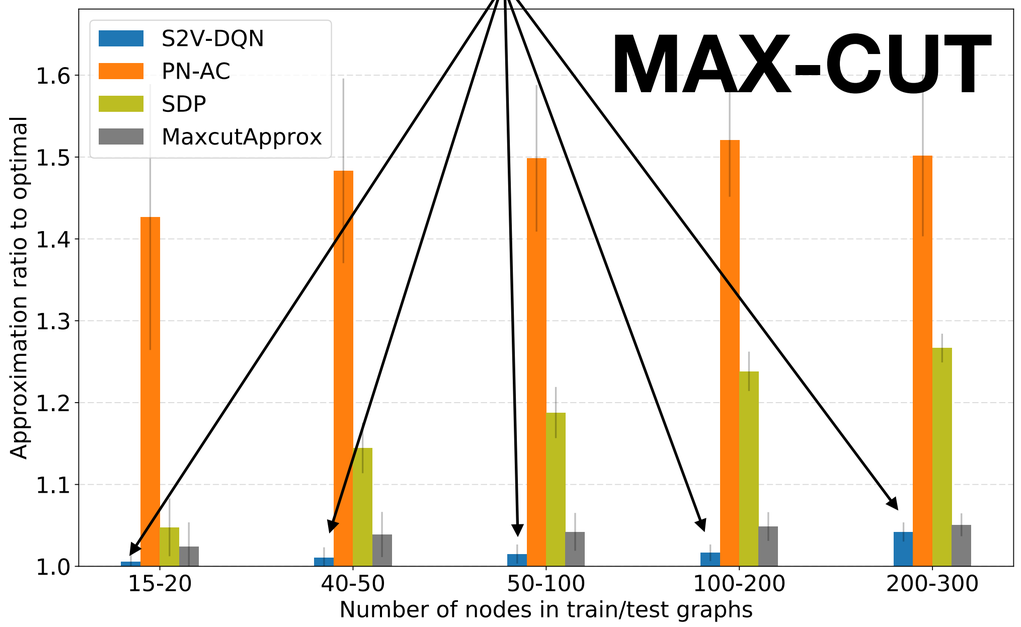
$$(y - \widehat{Q}(h(S_t), v_t; \Theta))^2$$
$$y = \gamma \max_{v'} \widehat{Q}(h(S_{t+1}), v'; \Theta) + r(S_t, v_t)$$

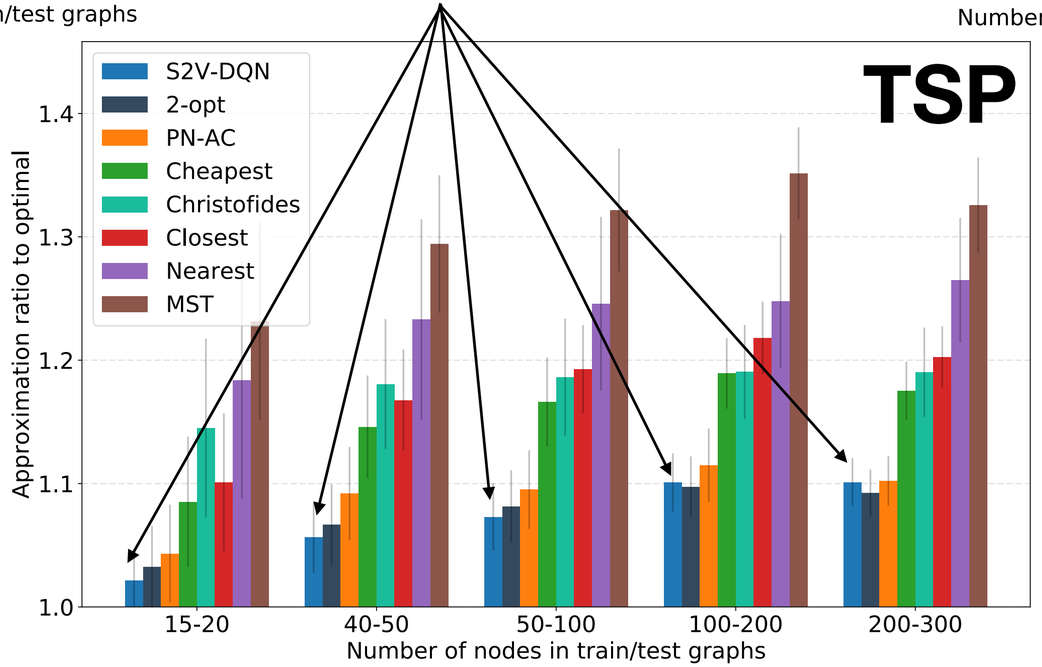
Overall Framework



Learning Greedy in Practice







Approximation Ratio

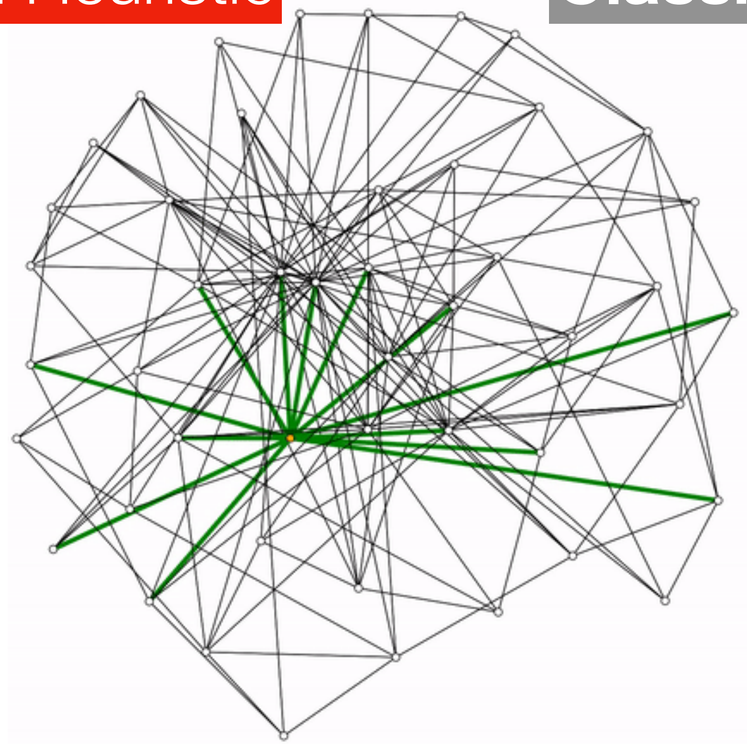
Code: https://github.com/Hanjun-Dai/graph_comb_opt

Data-Driven Algorithm Design

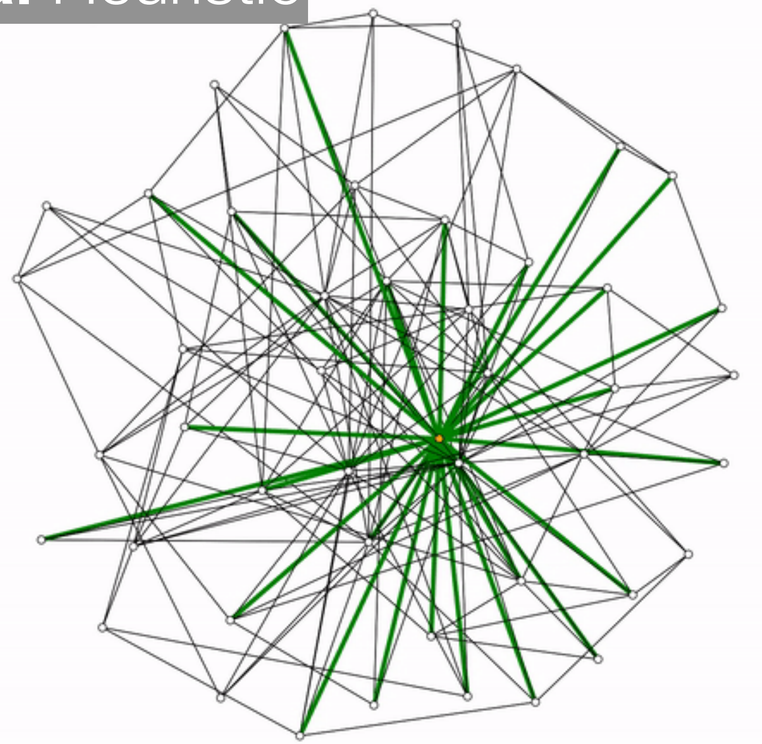
automatically **discovers**

novel search **strategies**

Learned Heuristic



Classical Heuristic



Minimum Vertex Cover

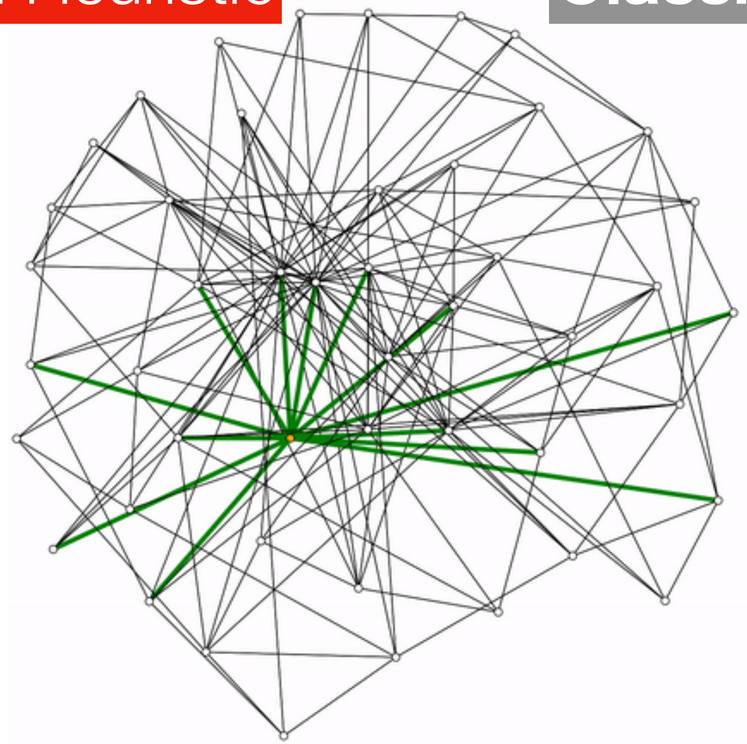
Find **smallest vertex subset** such that each **edge is covered**

Data-Driven Algorithm Design

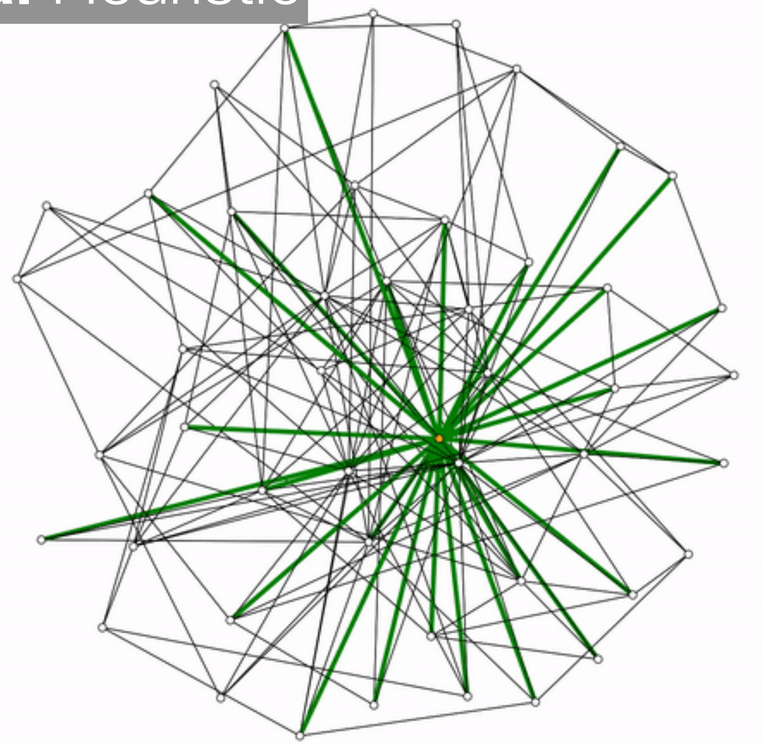
automatically **discovers**

novel search **strategies**

Learned Heuristic

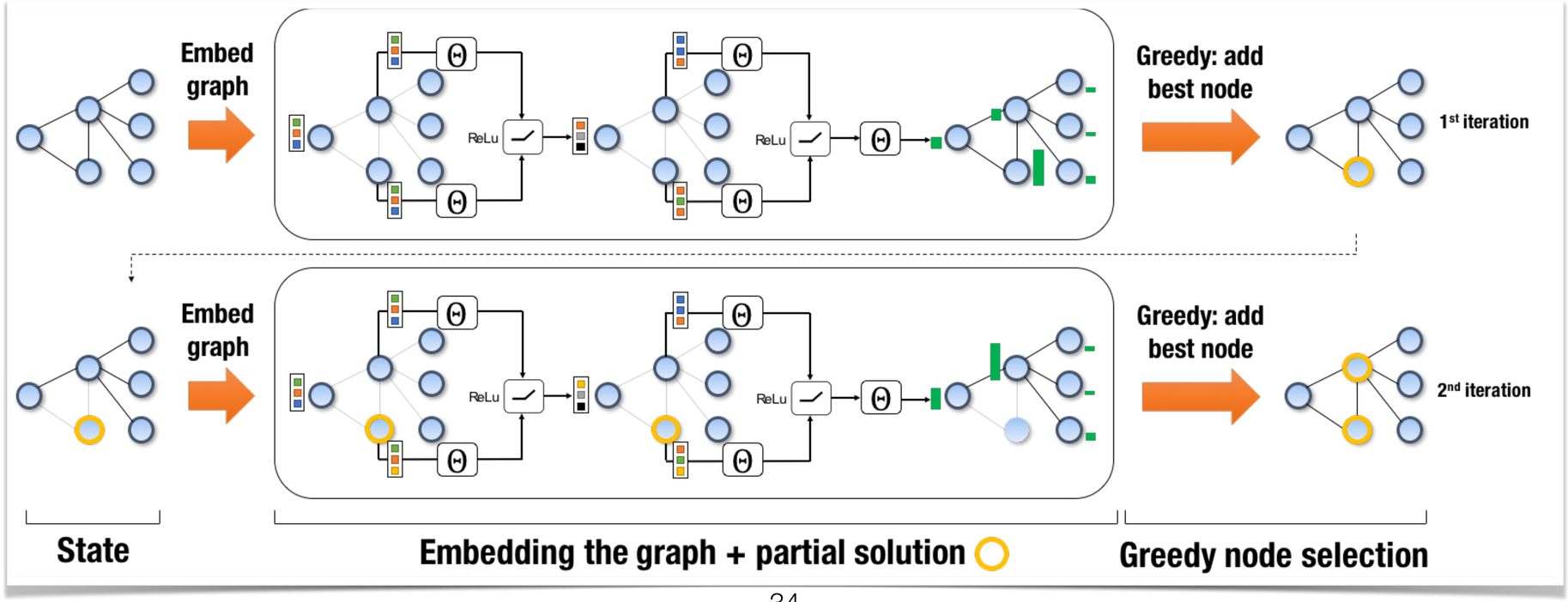


Classical Heuristic



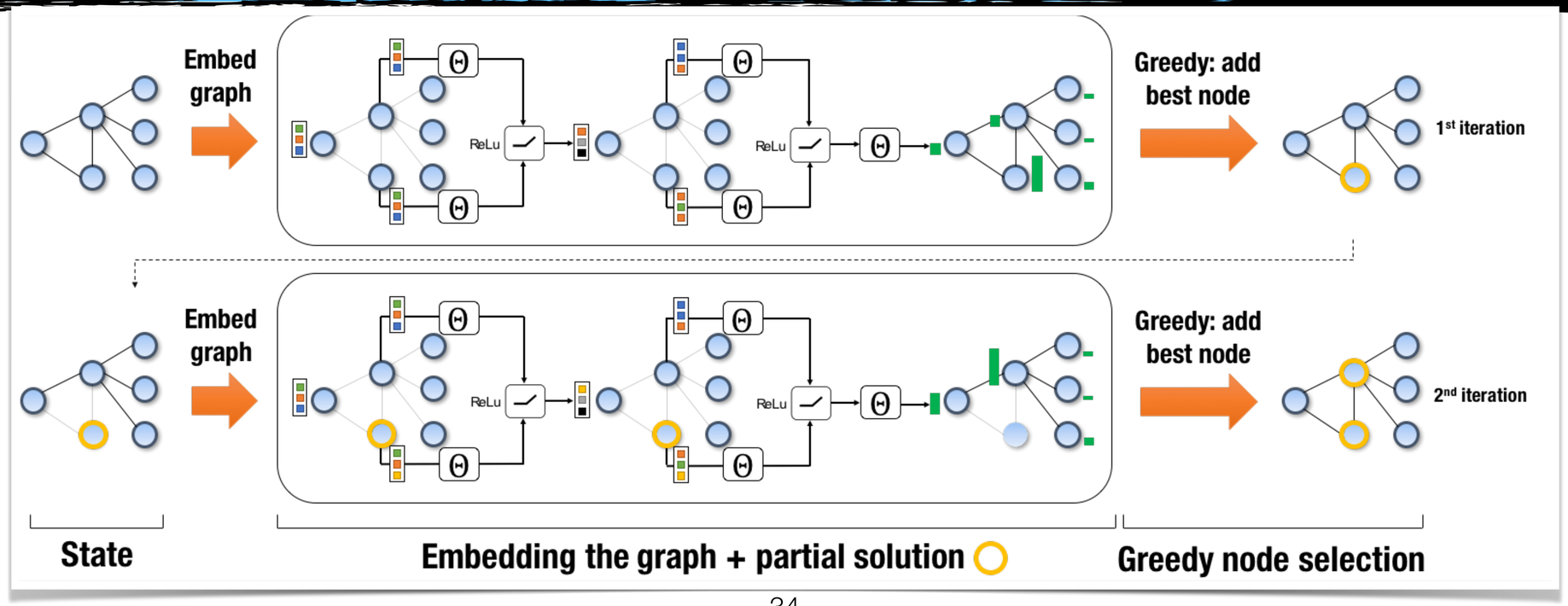
Minimum Vertex Cover

Find **smallest vertex subset** such that each **edge is covered**


Embed graph
ReLu
Greedy: add best node
1st iteration
Embed graph
ReLu
Greedy: add best node
2nd iteration
State
Embedding the graph + partial solution
Greedy node selection

Takeaways

- **Reinforcement Learning tailors greedy** search to your instances
- Learn **features jointly with** greedy **policy**
- **Human priors** encoded via (greedy) **meta-algorithm**
- **Interesting, novel strategies emerge**



Learning Heuristics: Recent progress

Published as a conference paper at ICLR 2019

ATTENTION, LEARN TO SOLVE ROUTING PROBLEMS!

Wouter Kool
University of Amsterdam
ORTEC
w.w.m.kool@uva.nl

Herke van Hoof
University of Amsterdam
h.c.vanhoof@uva.nl

Max Welling
University of Amsterdam
CIFAR
m.welling@uva.nl

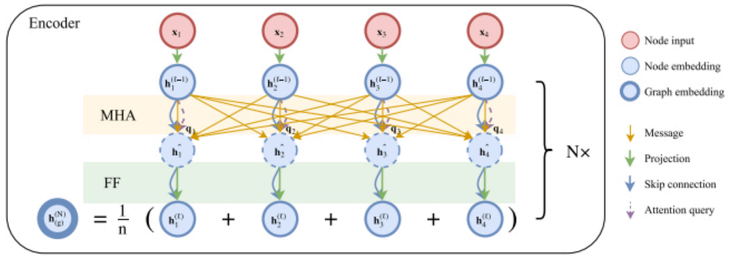


Figure 1: Attention based encoder. Input nodes are embedded and processed by N sequential layers, each consisting of a multi-head attention (MHA) and node-wise feed-forward (FF) sub-layer. The graph embedding is computed as the mean of node embeddings. Best viewed in color.

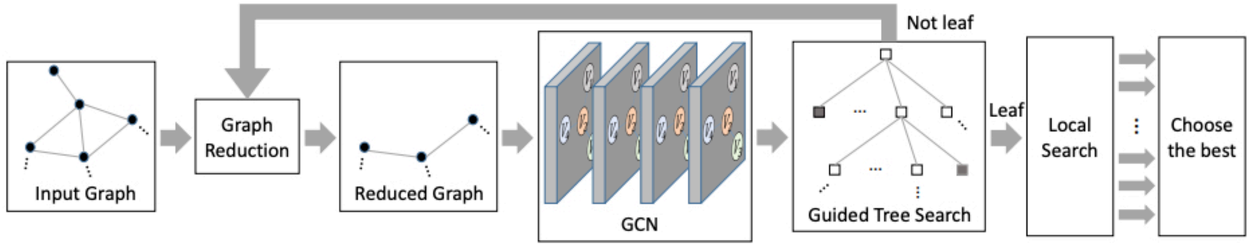
Learning Heuristics: Recent progress

Reinforcement Learning for Solving the Vehicle Routing Problem

Mohammadreza Nazari **Afshin Oroojlooy** **Martin Takáč** **Lawrence V. Snyder**
Department of Industrial and Systems Engineering
Lehigh University, Bethlehem, PA 18015
{mon314,afo214,takac,lvs2}@lehigh.edu

Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search

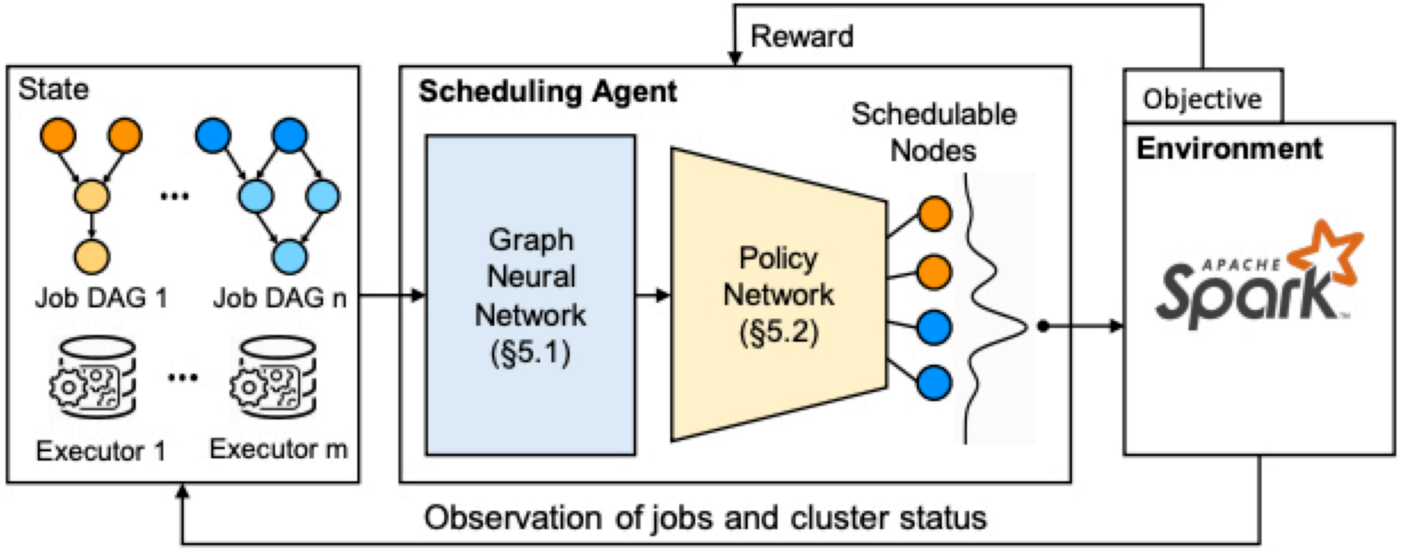
Zhuwen Li Intel Labs
Qifeng Chen HKUST
Vladlen Koltun Intel Labs



Learning Heuristics: Recent progress

Learning Scheduling Algorithms for Data Processing Clusters

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng*, Mohammad Alizadeh
MIT Computer Science and Artificial Intelligence Laboratory *Tsinghua University

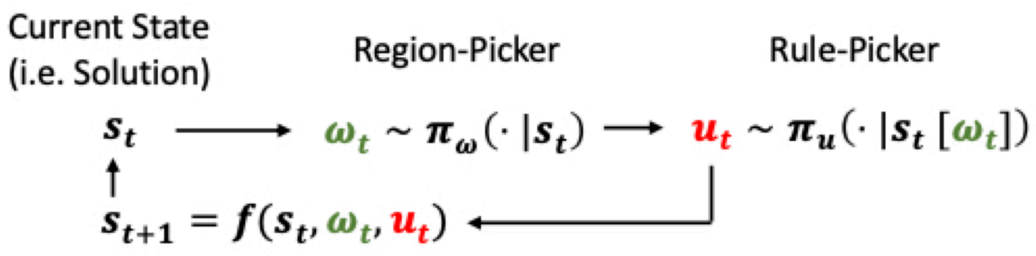


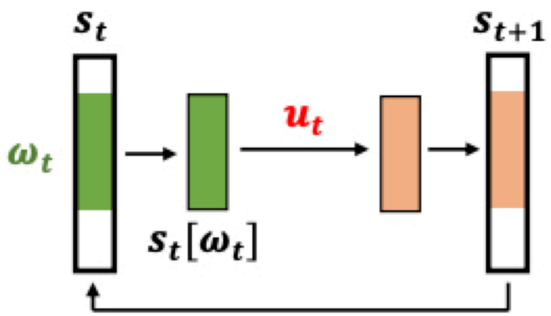
Learning Heuristics: Recent progress

Learning to Perform Local Rewriting for Combinatorial Optimization

Xinyun Chen *
UC Berkeley
xinyun.chen@berkeley.edu

Yuandong Tian
Facebook AI Research
yuandong@fb.com





Humans learn to **design algorithms**.

Machine Learning

Can **algorithms** “learn” to **design algorithms**?

Combinatorial Optimization

Machine Learning

Can **algorithms** “learn” to **design algorithms**?

Combinatorial Optimization

Machine Learning

Can **algorithms** “learn” to ~~**design**~~ **tailor** **algorithms**?

Combinatorial Optimization

Machine Learning

Can **algorithms** "learn" to ~~**design**~~ **tailor** **algorithms**?

Combinatorial Optimization

Yes!

Machine Learning

Can **algorithms** "learn" to ~~**design**~~ **tailor** **algorithms**?

Combinatorial Optimization

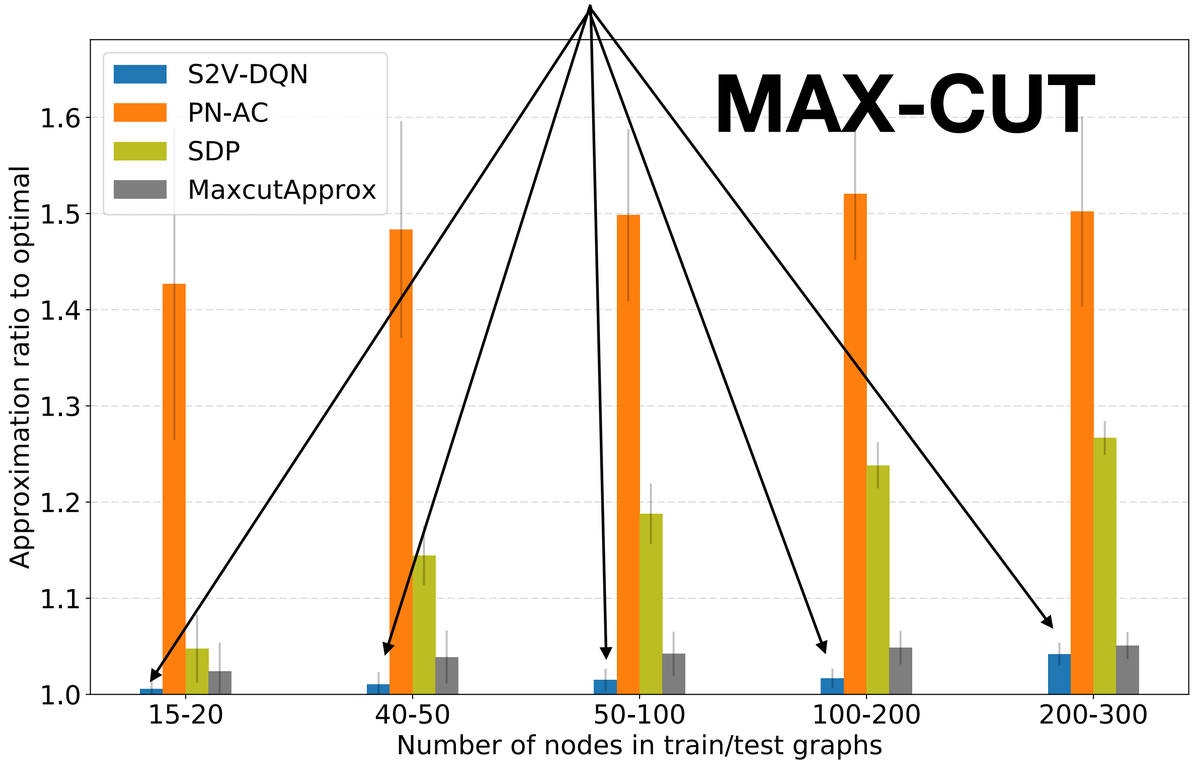
Yes!

ML complements human algorithms

ML fills in algorithm **details using data**

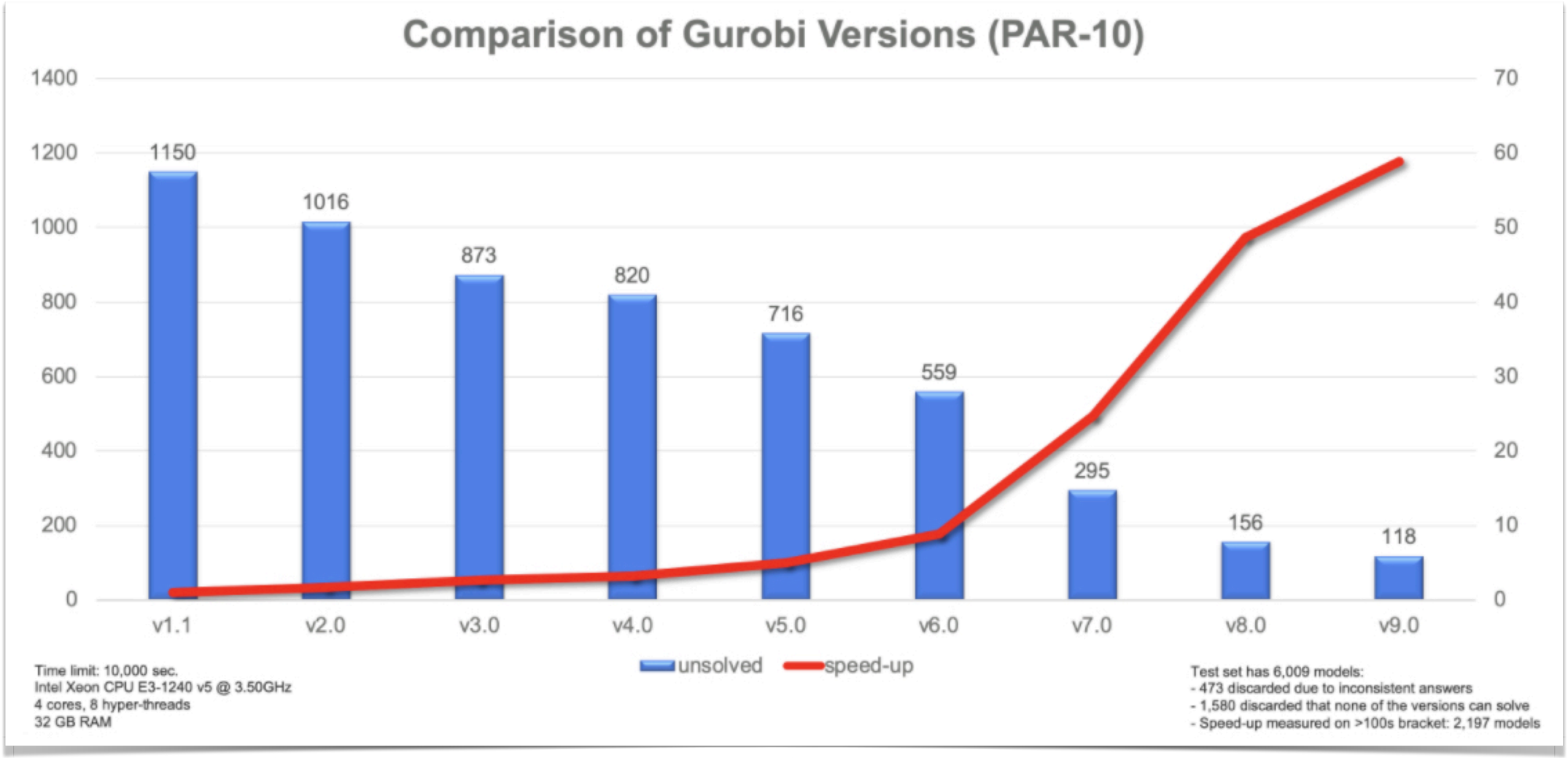
Beyond Heuristics…

Learned heuristic



Dai & Khalil et al., Learning Combinatorial Optimization Algorithms over Graphs. NeurIPS 2017.

Beyond Heuristics: Exact Solvers



https://www.gurobi.com/case_study/benchmarks/